28th TJIMO

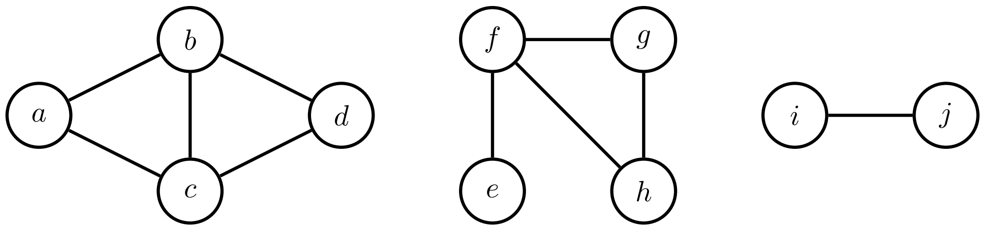
Alexandria, Virginia

Round: **Power**

This power round is divided into three sections. In the first section, the definitions from the Practice Power are reproduced for your convenience. In the second section, we explore shortest paths. In the third section, we explore Eulerian circuits. Sections 2 and 3 cover different material; you do not need section 2 to complete section 3. It might be wise to split up your team, with working on section 2 and the other team working on section 3.

1 Definitions from Practice Power

Definition 1. A *graph* $G(V, E)$ consists of a set of vertices V and a set of edges E.



Definition 2. An *edge* is a collection of exactly two vertices. In the above graph, there is an edge between a and b, so $\{a, b\}$ is an edge. Note that $\{a, b\}$ is the same edge as $\{b, a\}$.

Definition 3. A *set* is a collection of any number of objects. In the above graph, we have the set of vertices

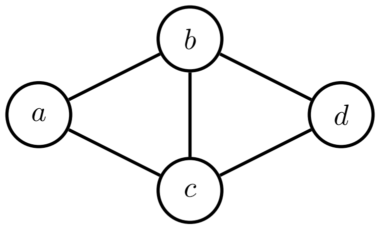
$$V = \{a, b, c, d, e, f, g, h, i, j\}.$$

We also have the set of edges

$$E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{b, d\}, \{a, c\}, \{e, f\}, \{f, g\}, \{g, h\}, \{h, f\}, \{i, j\}\}.$$

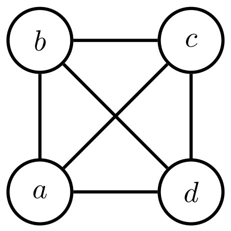
Definition 4. Given a set A, the cardinality $|A|$ denotes the number of elements in A.

Definition 5. The *degree* of a vertex is the number of edges that connect that vertex to another. For vertex b in the following graph, we write $deg(b) = 3$, meaning the degree of b is 3.

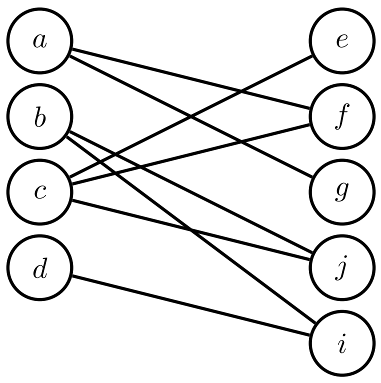


Definition 6. A *clique* is a graph such that there is exactly one edge covering every pair of distince vertices in the graph. We denote a clique of n vertices as K_n. Pictured is a clique of 4 vertices, or K_4.

Time limit: 45 minutes.



Definition 7. A *bipartite graph* is a graph whose vertices can be divided into two disjoint, or nonoverlapping, sets, S and T, such that every edge in the graph connects one vertex in S to one vertex in T. The following is an example of a bipartite graph:



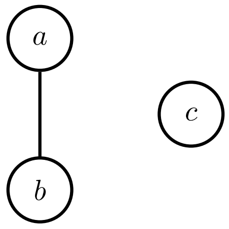
We can split the vertices into two sets: $S = \{a, b, c, d\}$ and $T = \{e, f, g, h, i\}$. Every edge in the graph connects a vertex in S to one in T. No edge connects two vertices from S, and no edge connects two vertices from T.

2 Shortest Paths

Graphs represent connections between different objects. We might, for example, have different cities connected in some way. It takes a certain amount of time to get in between different neighboring cities. Perhaps to travel from D.C. to St. Louis, we must either stop at Chicago or Cincinnati along the way. It might be faster to travel through Cincinnati, but it might be cheaper to stop at Chicago. Either way, we need to devise some kind of system to determine for us the best way to travel between vertices in our graph, through following some kind of *path*.

2.1 Introduction to Paths

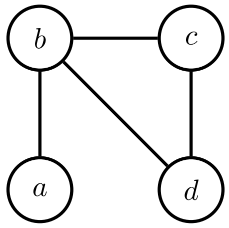
Definition 8. A *walk* is a sequence of vertices such that every two consecutive vertices in the walk are connected by an edge.
Definition 9. A *path* is a walk in which all edges in the path are distinct and all the vertices are distinct.
Definition 10. A graph is called *connected* when there is a path between every pair of vertices. For example, the following graph is not connected, since there is no path from a to c.



Problem 1. What is the minimum number of edges needed for a graph with n vertices to be connected? Remember to explain your answer!

From now on, all graphs we consider will be connected.

Problem 2. In the following graph, answer the questions below.



(a) Is $\{b, c\}$ a path? That is, is the walk that starts at b and travels to c a path?

(b) Is $\{b, c, d\}$ a path? That is, is the walk that starts at b, goes to c, and ends at d a path?

(c) Is $\{a, c, b\}$ a path?

(d) Is $\{b, c, b\}$ a path?

(e) Is $\{b, c, d, b\}$ a path?

(f) Is $\{b\}$ a path?

If your answer for any of these was "no," explain why.

2.2 Dijkstra's Shortest Path Algorithm

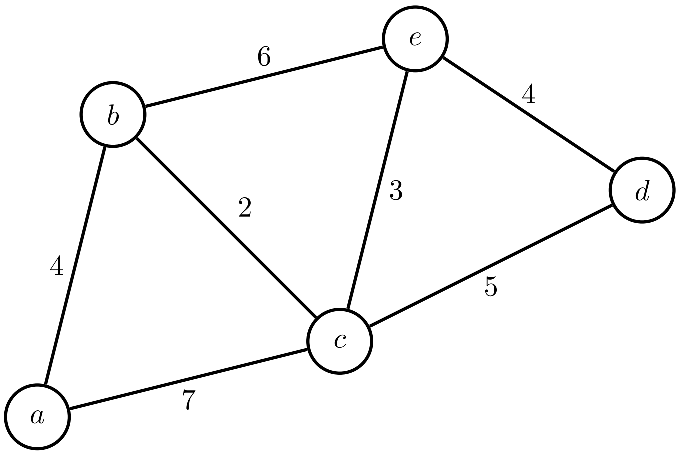
In this subsection we explore an algorithm to compute the shortest, or cheapest, path from one vertex to another in a graph. Thus, we'll want to assign some kind of value to each edge. For example, it might be the distance between two vertices. We'll call this value the *weight*.

Definition 11. We assign to each edge $\{u, v\}$ a nonnegative *weight* $w(u, v) \geq 0$. This is simply a number associated with the edge. For example, it might represent the distance between u and v or the cost to travel between u and v.

Our main goal is to find a path from one vertex to another such that the sum of the weights along the path is minimized. To travel along an edge in the path, we have to pay the weight of that edge.

Definition 12. Given a graph G, the *shortest path* $\{v_0, v_1, v_2, v_3, \cdots, v_{m-2}, v_{m-1}, v_m\}$ from vertex v_0 to vertex v_m is the path that minimizes the sum

$$w(v_0, v_1) + w(v_1, v_2) + w(v_2, v_3) + \cdots + w(v_{m-2}, v_{m-1}) + w(v_{m-1}, v_m).$$



In the above graph, the shortest path from a to e is a, b, c, e. The shortest, or cheapest, distance from a to e is then 9. No other path can do better.

Problem 3. Determine the shortest path and shortest-path length of each vertex from a. List the vertices in increasing order of the distance from a to that vertex.

Definition 13. A *source* is a vertex from which paths begin. In the above problem, a was the source. Dijkstra's algorithm is a single-source algorithm, which means it computes the shortest path from one source to all other vertices.

Problem 4. Let u be a vertex immediately prior to vertex v along the shortest path from source s to v. Show that the shortest path from s to v must pass along a shortest path to u before traversing one final edge from u to v.

Dijkstra's algorithm involves keeping track of a best known distance, which we'll call $dist(v)$ for each vertex v. Initially, $dist(v) = \infty$ for all vertices v, except for the source s of our graph, for which $dist(s) = 0$.

Then, if we were to run the algorithm on the example graph from before, the $dist$ of each vertex is as follows, at the very beginning:

vertex v	$dist(v)$
a	0
b	∞
c	∞
d	∞
e	∞

As we explore our graph, $dist(v)$ decreases, until $dist(v)$ represents the actual shortest-path distance from a source to v.

The way we explore the graph is we visit vertices one by one, in the order of closeness to the source, as you have computed in Problem 3. Therefore, the first vertex we visit is the source itself.

We only visit a vertex v if we know for sure that the actual shortest-path distance to v matches what we have stored in $dist(v)$. When we visit v, we take a look at all vertices u which share an edge with v. If $dist(v)+w(v,u) < dist(u)$, then we change the value of $dist(u)$ to be $dist(v)+w(v,u)$. In other words, we have explored enough of our graph to discover a path through v to u that is shorter than the one known up to that point. This means that $dist(v)$, which is the best known distance thus far, also represents the best path to v through all visited vertices.

Notice how, in the table below, $dist(b)$ and $dist(c)$ are updated, as they are neighbors of a, which was just visited.

vertex v	$dist(v)$	visited?
a	0	yes
b	4	no
c	7	no
d	∞	no
e	∞	no

The vertex that has the minimum $dist$ that we haven't visited yet is b. Notice how $dist(c)$ and $dist(e)$ are updated in the table below, as they are neighbors of b.

vertex v	$dist(v)$	visited?
a	0	yes
b	4	yes
c	6	no
d	∞	no
e	10	no

$dist(b) = 4$, which agrees with the actual shortest path distance to b.

Problem 5. What is the next vertex we should visit? (Hint: which unvisited vertex has the minimum $dist$?) Copy the table in your answer and update it so that three vertices are visited. Does $dist$ of this vertex agree with the actual shortest path distance to it, as you computed in Problem 3?

What you should notice is that $dist(v)$ is the actual shortest distance to v for all visited vertices v.

Problem 6. Suppose that we have just visited k vertices, where $k > 0$ is some positive integer. Furthermore, suppose that these k vertices are the k closest of all the vertices to the source. Show that, of all the unvisited vertices, the vertex v with the minimum $dist(v)$ is such that $dist(v)$ is equal to the actual shortest distance from the source to v. In other words, show that we can visit this vertex and make it the $(k+1)$-th visited vertex.

The previous problem allows us to continue to visit the closest unvisited vertex, until we have visited all the vertices. Once we finish visiting all the vertices, $dist(v)$ will represent the shortest distance from the source to v. Dijkstra's algorithm in its entirety is outlined below.

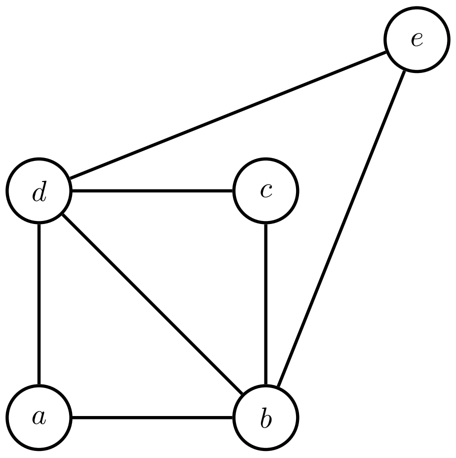
Algorithm 1 Dijkstra

```
for all vertices $v$ do
    $dist(v) \leftarrow \infty$                          ▷ This means set $dist(v)$ to be $\infty$.
    $visited(v) \leftarrow 0$                         ▷ This means set $visited(v)$ to be 0.
    $prev(v) \leftarrow -1$          ▷ This allows us to trace our path backwards to the source.
$dist(src) \leftarrow 0$                         ▷ Initialize the value of $dist$ for the source.
while there exists a vertex $v$ such that $visited(v) = 0$ do
    Let $v$ be the vertex such that $visited(v) = 0$ that minimizes $dist(v)$
    $visited(v) \leftarrow 1$                                        ▷ Visit the vertex.
    for all neighbors $u$ of $v$ do
        if $visited(u) = 0$ then
            $alt \leftarrow dist(v) + weight(v, u)$
            if $alt < dist(u)$ then
                $dist(u) \leftarrow alt$      ▷ If we found a different path, update the value of $dist$.
                $prev(u) \leftarrow v$
```

Dijkstra's algorithm lets us comput the shortest or cheapest path from one vertex to another extremely quickly – roughly on the order of the number of edges. This means that even in a graph with 100,000 vertices and 100,000 edges a computer program can compute the shortest distance from one vertex to another in a fraction of a second! Perhaps in high school, you will code this up yourself and see it in action.

3 Eulerian Circuits

Definition 14. An *Eulerian circuit*, named after the Swiss mathematician Leonhard Euler, is a walk that begins and ends on the same vertex and traverses every edge in the graph exactly once. In the following graph, one possible Eulerian circuit is $\{a, b, c, d, b, e, d, a\}$.

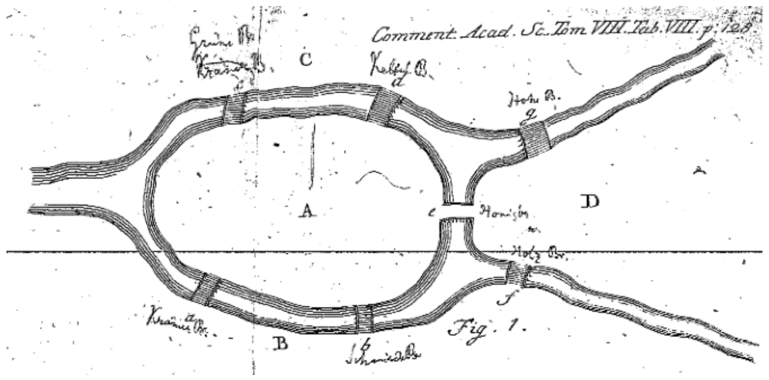


Notice how each edge is traversed exactly once. Not once in one direction and once in the other – once in total!

Problem 7. Can any connected graph have an Eulerian circuit? If not, draw a graph that does not have an Eulerian circuit.

Problem 8. Show that, if a connected graph has an Eulerian circuit, the degree of every vertex must be even. Remember that the degree of a vertex is the number of edges adjacent to it.

Problem 9 (Seven Bridges of Königsberg)**.** The port city of Königsberg in Prussia (now Kaliningrad, Russia) lies on two sides of a large river. In addition, there are two islands within the river also part of the city. Seven bridges connect the four land masses, as pictured. Is it possible to walk through the city but cross every bridge exactly once? Euler famously solved this problem in 1736; this is his drawing of Königsberg, from his paper "Solutio problematis ad geometriam situs pertinentis."



Problem 10. In connected graph G, every vertex has even degree. Suppose we began a walk at vertex v. We get stuck if we enter a vertex that has no outgoing edge we haven't yet traversed. Show that it is impossible to get stuck at a vertex other than v, and use this fact to show that any connected graph G, with all vertices having even degree, has an Euler circuit.

That concludes the Power Round. We hope you learned something cool!